US-PAT-NO: 6446203

DOCUMENT-IDENTIFIER: US 6446203 B1

TITLE: Method and system for selecting from multiple boot code images to be loaded in a data processing system

---------- KWIC ---------

Abstract Text - ABTX (1):
A computer system including a processor, a system memory, and a boot code storage device. The system memory is connected to the processor and is suitable for storing processor data and instructions. The boot code storage device includes an image selection indicator for indicating which of multiple boot code images are to be loaded. The computer system further includes means for initiating a boot sequence stored on the boot code storage device. The boot sequence selects from **first and second boot images** based upon the state of the image selection indicator and loads the selected image into the system memory in response to a boot event. In one embodiment, the image selection indicator is in an initial state until the boot code sequence successfully loads a boot image. The image selection indicator is set to a value indicative of the loaded image when one of the boot images is successfully loaded. The boot code storage sequence attempts, in the preferred embodiment, to load the previously load boot image when the system detects a setting of the image selection indicator indicative of a previously loaded boot image. In the preferred embodiment, the boot code sequence responds to a specified user input sequence by presenting the user with a configuration screen suitable for altering the value of the image selection indicator such that the user may later the boot image that will be loaded during a subsequent execution of the boot sequence. In one embodiment, the **first boot image** is a BIOS based boot image and the second image is a network based (non-BIOS) boot image. The boot code storage device is preferably a flash memory device.

Brief Summary Text - BSTX (5):

Computer systems generally require some manner of boot procedure to achieve a functional state after a system reset, a system wake up, or a power-on event. The boot procedure typically includes executing a boot sequence stored on a nonvolatile memory of the system. The boot sequence is responsible for loading a boot image into system memory. Historically, a computer system's boot device was suitable for loading only a single boot image. With a conventionally designed machine, an end user who wishes to load a different boot image than the boot image stored on the system's boot device is required to obtain an second boot or reprogram the existing boot device, thereby losing the original boot image in the process. Manufacturers of conventionally designed systems must stock multiple types of boot devices to meet the various needs of their customers. Customers requiring a **first type of boot image** will require a first boot device while customers requiring a **second type of boot image** will require a second boot device thereby creating potential manufacturing concerns including order tracking and inventory management for the manufacturer.

Moreover, the differing requirements of various customers prevents the manufacturer from being able to build, stock, and sell a single machine that can meet the needs of all of its customers (assuming that, aside from the boot image, the different customers have the same computing requirements). In addition, the number of users who require the option of being able to boot from multiple boot images is increasing. Such users face currently face the formidable task of reprogramming or physically replacing the boot device in their systems each time they need to switch images. Accordingly, for a number of reasons, it is would be highly desirable to offer a system capable of easily switching between multiple boot images.

Brief Summary Text - BSTX (7):

Broadly speaking the present invention contemplates a computer system and an accompanying boot code storage device. According to the invention, the computer system includes a processor, a system memory, and the boot code storage device. The system memory is connected to the processor and is suitable for storing processor data and instructions. The boot code

storage device includes an image selection indicator for indicating which of multiple boot code images are to be loaded into the system memory. The computer system further includes means for initiating a boot sequence stored on the boot code storage device. The boot sequence selects from **first and second boot images** based upon the state of the image selection indicator and loads the selected image into the system memory in response to a boot event. In one embodiment, the image selection indicator is in an initial state until the boot code sequence successfully loads one of the boot images. The image selection indicator is set to a value indicative of the loaded image when one of the boot images is successfully loaded. The boot code storage sequence attempts, in the preferred embodiment, to load a previously loaded boot image when the system detects a setting of the image selection indicator indicative of a previously loaded boot image. In other words, the system has a preference to load a previously loaded boot image. In the preferred embodiment, the boot code sequence responds to a specified user input sequence by presenting the user with a configuration screen suitable for altering the value of the image selection indicator such that the user may alter the boot image that will be loaded during a subsequent execution of the boot sequence. In one embodiment, the **first boot image** is a BIOS based boot image and the second image is a network based (non-BIOS) boot image. The boot code storage device is preferably a flash memory device.

Detailed Description Text - DETX (7):

The present invention contemplates a boot code storage device capable of selecting a boot image from two or more boot images on the boot code storage device and storing the selected boot image to system memory 106. Turning to FIG. 2, the depicted embodiment of boot code storage device 120 according to the invention includes **first and second boot code images** 122 and 124 respectively as well as a boot image selection indicator 126. In response to a boot event such as powering on computer system 100, pushing a reset button on the system cabinet, or a network wake-up event, computer system 100

initiates a
boot code sequence stored on boot code storage device 100. The boot code
sequence determines the state of image selection indicator 126. In one
embodiment, image selection indicator 126 is in an initial state when the
computer system is powered on for the first time and remains in the initial
state until one of the available boot images has been loaded successfully.
When image selection indicator 126 is detected by the boot sequence as being in
the initial state, the boot sequence will attempt to load each of the available
boot images on boot codes storage device 120 until one of the boot images loads
successfully. If the boot code sequence succeeds in loading one of the
available boot images and computer system 100 achieves an operational state
running under the loaded boot image, image selection indicator 126 is altered
to indicate the boot image that was successful. Upon subsequent boot events,
the boot code sequence will detect the altered state of image selection
indicator 126 and proceed to load the boot image indicated by indicator 126.
In this manner, once a particular boot image has been loaded successfully, the
presumption is that the system will run using this boot image thereafter.
Although the depicted embodiment indicates image selection indicator 126 as
contained with boot code storage device 120, other embodiments may include
non-volatile storage external to boot code storage device such as an external
EPROM, an additional flash memory device, or a battery backed CMOS storage
device in which image selection indicator 126 is stored.

Detailed Description Text - DETX (8):
The boot image presumption indicated by image selection indicator 126 may be
altered by a user of computer system 100 when it is desirable to boot computer
system 100 with a different boot image. In one embodiment, the user can alter
the state of image selection indicator 126 through a "set-up" menu invoked by
entering a specified keyboard or other input sequence during the boot time
frame. Thus, a user who desires to boot computer system 100 with **second boot image** 124 on a system whose image selection indicator 126 indicates **first boot image** 122 as the default boot image may modify indicator 126 by booting
computer system 100 and entering the appropriate keyboard sequence

while the
boot sequence is executing. Computer system 100 will respond to the input
sequence by presenting the user with a configuration screen via a display
terminal (not depicted). The configuration menu will include an entry for
image selection indicator 126 and facilities for modifying the value or setting
of indicator 126 and for re-executing the boot code sequence with the new value
of indicator 126.

Detailed Description Text - DETX (9):
Boot code storage device 120, as depicted in FIG. 2, includes **first and second boot images** 122 and 124 respectively. In one embodiment of the invention first boot 122 is a BIOS based boot image and **second boot image** 124
is a non-BIOS boot image referred for purposes of this disclosure as a network-based boot image. BIOS based boot images and operating systems are
well known in the field of personal computers. The BIOS provides micro code at
the lowest level that controls the I/O device hardware of system 100 such as
disk drives. Detailed BIOS information may be found in Croucher, Que's BIOS
Companion (MacMillan 1998), incorporated by reference herein. The network
based boot image contemplated herein is suitable for using with loading an
operating system such as JavaOS. Network based boot images and operating
systems as contemplated herein are relatively small (compared to BIOS based
boot images and operating systems) platform independent operating systems
adapted to execute applications directly on hardware platforms without requiring any other host operating system. In one embodiment, network based
boot image 124 is recognizable by its executable and linking format (ELF)
compliant header section. Network based boot images and operating systems are
specifically designed to support distributed processing across a variety of
products in a multi-platform, thin-client environment.

Claims Text - CLTX (1):
1. A computer system comprising: a processor; a system memory connected to
the processor and suitable for storing processor data and instructions; a boot
code storage device including an image selection indicator; and means for

initiating a boot sequence stored on the boot code storage device, wherein the boot sequence selects from **first and second boot images** based upon the state of the image selection indicator and loads the selected image into the system memory in response to a boot event, wherein the **first boot image** comprises a BIOS boot image suitable for use with BIOS-based operating systems and wherein the **second boot image** comprises a network boot image suitable for use with a network operating system.

Claims Text - CLTX (2):
2. The computer system of claim 1, wherein the **first and second boot code images** are stored on the boot code storage device.

Claims Text - CLTX (7):
7. The computer system of claim 1, wherein the **second boot image** includes an Executable and Linking Format (ELF) compliant header section.

Claims Text - CLTX (8):
8. The computer system of claim 7, wherein the **second boot image** comprises a JavaOS boot image.

Claims Text - CLTX (9):
9. A boot code storage device for use with a computer system, wherein the boot code storage device is configured with **first and second boot code images** and an image selection indicator, and further configured with a boot code storage sequence that selects the **first or second boot code images,** responsive to the value of the image selection indicator, and loads the selected image into a system memory of a computer system, wherein the **first boot code image** comprises a BIOS-based boot image suitable for use with a BIOS-based operating system and the **second boot code image** comprises a network boot image suitable for use with a network operating system.

Claims Text - CLTX (11):
11. The storage device of claim 9, wherein the image selection indicator is in an initial state until the boot code sequence successfully loads either the

first or second boot image.

Claims Text - CLTX (15):
15. The storage device of claim 9, wherein the **second boot image** includes
an Executable and Linking Format (ELF) compliant header section.

Claims Text - CLTX (16):
16. The storage device of claim 15, wherein the **second boot image** comprises
a JavaOS boot image.

Current US Original Classification - CCOR (1):
713/2

US-PAT-NO: 5842011

DOCUMENT-IDENTIFIER: US 5842011 A

TITLE: Generic remote boot for networked workstations by creating **local bootable code image**

---------- KWIC ---------

TITLE - TI (1):
Generic remote boot for networked workstations by creating **local bootable code image**

Drawing Description Text - DRTX (12):
FIG. 14 is a memory map diagram showing the state of the PC client memory just **prior to booting from the LAD disk image** during the generic remote boot procedure of this invention.

Claims Text - CLTX (3):
copying an image of the bootable code over the network to the local data processing system to create a **local bootable code image** stored in the local data processing system; and

Claims Text - CLTX (4):
booting the local data processing system from the **local bootable code image** stored in the local data processing system.

Claims Text - CLTX (14):
copying an image of the bootable code over the network to the local data processing system to create a **local bootable code image** stored in the local data processing system by copying an image of the remote disk over the network to the local data processing system to create a local disk image stored in the local data processing system; and

Claims Text - CLTX (15):
booting the local data processing system from the **local bootable code image** stored in the local data processing system by booting the local data

processing
system from the local disk image.

Claims Text - CLTX (48):
copying an image of the bootable code over the network to the local data
processing system to create a **local bootable code image** stored in the local
data processing system; and

Claims Text - CLTX (49):
booting the local data processing system from the **local bootable code image**
stored in the local data processing system.

Claims Text - CLTX (53):
copying an image of the bootable code over the network to the local data
processing system to create a **local bootable code image** stored in the extended
memory of the computer; and

Claims Text - CLTX (54):
booting the computer from the **local bootable code image** stored in the
extended memory.

Claims Text - CLTX (57):
copying an image of the bootable code over the network to the local data
processing system to create a **local bootable code image** stored in the local
data processing system;

Claims Text - CLTX (59):
booting the local data processing system from the **local bootable code image**
stored in the local data processing system;

Claims Text - CLTX (155):
copying means for copying an image of the bootable code over the network to
the local data processing system to create a **local bootable code image** stored
in the local data processing system; and

Claims Text - CLTX (156):
booting means for booting the local data processing system from the **local
bootable code image** stored in the local data processing system.

Claims Text - CLTX (167):
booting means for booting the local data processing system from the **local bootable code image** from the local disk image.

Claims Text - CLTX (170):
copying means for copying an image of the bootable code over the network to the local data processing system to create a **local bootable code image** stored in the local data processing system; and

Claims Text - CLTX (171):
booting means for booting the local data processing system from the **local bootable code image** stored in the local data processing system.

Claims Text - CLTX (187):
copying means for copying an image of the bootable code over the network to the local data processing system to create a **local bootable code image** stored in the local data processing system; and

Claims Text - CLTX (188):
booting means for booting the local data processing system from the **local bootable code image** stored in the local data processing system.

Claims Text - CLTX (206):
copying means for copying an image of the bootable code over the network to the computer to create a **local bootable code image** stored in the extended memory of the computer; and

Claims Text - CLTX (207):
booting means for booting the computer from the **local bootable code image** stored in the computer.

Claims Text - CLTX (210):
copying means for copying an image of the bootable code over the network to the local data processing system to create a **local bootable code image** stored in the local data processing system; and

Claims Text - CLTX (211):
booting means for booting the local data processing system from the **local bootable code image** stored in the local data processing system;

Claims Text - CLTX (219):
copying means for copying an image of the bootable code over the network to the workstation to create a **local bootable code image** stored in the workstation; and

Claims Text - CLTX (220):
booting means for booting the workstation from the **local bootable code image** stored in the workstation.

Claims Text - CLTX (223):
copying an image of the bootable code over the network to the local data processing system to create a **local bootable code image** stored in the local data processing system;

Claims Text - CLTX (225):
booting the local data processing system from the **local bootable code image** stored in the local data processing system while the network communications are inactive.

Current US Original Classification - CCOR (1):
713/2

US-PAT-NO: 5933631

DOCUMENT-IDENTIFIER: US 5933631 A
See image for Certificate of Correction

TITLE: Dynamic boot filesystem selection

---------- KWIC ---------

Claims Text - CLTX (2):
loading a **first boot image** which contains a dynamic boot filesystem, using
the computer's firmware;

Claims Text - CLTX (4):
loading a **second boot image** which contains the operating system, using the
computer's firmware, after said storing step;

Claims Text - CLTX (17):
8. The method of claim 1 wherein the **first and second boot images** are
provided on a single media which is scanned twice.

Claims Text - CLTX (18):
9. The method of claim 1 wherein the **first boot image** contains instructions
which release control of the computer back to the firmware after storing the
dynamic boot filesystem in the protected space.

Claims Text - CLTX (19):
10. The method of claim 1 wherein the **second boot image** contains
instructions which retrieve the dynamic boot filesystem from the protected
space during the boot sequence.

Claims Text - CLTX (21):
the **first boot image** is provided on a first media; and

Claims Text - CLTX (22):
the **second boot image** is provided on a second media separate from the first
media.

Claims Text - CLTX (26):
the firmware automatically seeks the **second boot image** after the

first boot image has been loaded.

Claims Text - CLTX (32):
media adapted to be read by at least one of said input devices, having a **first boot image** containing a dynamic boot filesystem and instructions for (i) storing said dynamic boot filesystem in said memory device and (ii) thereafter returning control to said firmware means, and having a **second boot image** containing said operating system and instructions for retrieving said stored dynamic boot filesystem.

Claims Text - CLTX (33):
15. The device of claim 14 wherein said **second boot image** further contains a default boot filesystem and instructions for loading said default boot filesystem if said dynamic boot filesystem cannot be retrieved.

Claims Text - CLTX (36):
said **second boot image** further contains a hardware-dependent PAL in said dynamic boot filesystem.

Claims Text - CLTX (37):
17. The device of claim 14 wherein the **second boot image** further contains hardware-dependent device drivers.

Claims Text - CLTX (38):
18. The device of claim 14 wherein said **first boot image** further contains instructions for storing control information in said memory device which identifies a location of said stored dynamic boot filesystem.

Claims Text - CLTX (40):
a first media having said **first boot image**; and

Claims Text - CLTX (41):
a second media separate from said first media, having said **second boot image**.

Current US Original Classification - CCOR (1):
713/2

US-PAT-NO: 6594786

DOCUMENT-IDENTIFIER: US 6594786 B1

TITLE: Fault tolerant high availability meter

---------- KWIC ---------

Brief Summary Text - BSTX (6):

The vast majority of servers are supplied with conventional cost-effective availability features, such as backup. Enhanced hardware technologies have been developed to improve availability in excess of 95%, including automatic server restart (ASR), uninterruptable power supplies (UPS), backup systems, hot swap drives, RAID (redundant array of inexpensive disks), duplexing, manageable ECC (**error** checking and correcting), memory scrubbing, redundant fans, and hot swap fans, **fault-resilient** processor **booting,** pre-failure alerts for system components, redundant PCI (peripheral component interconnect) I/O (input/output) cards, and online replacement of PCI cards. The next segment of server usage is occupied by high-availability servers with uptimes in excess of 99.9%. These servers are used for a range of needs including internet services and client/server applications such as database management and transaction processing. At the highest end of the availability spectrum are systems that require continuous availability and which cannot tolerate even momentary interruptions, such as air-traffic control and stock-floor trading systems.

Brief Summary Text - BSTX (17):

According to a preferred embodiment of the present invention, a fault tolerant method of monitoring one or more computers for availability may include generating an event when a computer system **detects a change in its status** that affects availability; transmitting the event from the computer system to a central repository; and periodically re-transmitting the event if a receipt confirmation message is not received from the central repository. The computer system may store the event in a local repository located on

the
computer system before transmitting the event to the central repository. If a
receipt confirmation message is not received from the central repository, the
event is held in a queue for re-transmission at a later time. If the computer
system receives a status request from the central repository, in addition to
reporting status, the computer system will transmit the events held in the
queue.

Brief Summary Text - BSTX (18):
The present invention also includes a fault tolerant method of monitoring
one or more computers for availability, where the method may include generating
an event containing a sequence number when a computer system **detects a change
in its status** that effects availability; transmitting the event from the
computer system to a central repository; comparing the sequence number of the
event with a next expected sequence number computed from reading the central
repository; and synchronizing events between the computer system and the
central repository if the sequence number does not match the next expected
sequence number. A copy of each event may be maintained in a local repository
on the computer system. If the sequence number matches the next expected
sequence, the events and sequence numbers are stored in the central repository.
If the sequence number is greater than the next expected sequence number, the
central repository requests the missing events from the computer system. If
the sequence number is less than the next expected sequence number, the central
repository determines whether the event has already been received. If the
event has already been received, the event is discarded. If the event has not
already been received, the computer system has lost events and the central
repository sends the missing events to the computer system.

Detailed Description Text - DETX (20):
In the preferred embodiment, the event monitoring service 40 is a system
monitoring application designed to facilitate real-time monitoring and error
detection for enterprise systems, such as cluster C. In an alternative

embodiment, the real-time monitoring of cluster resources could be designed into the HA agent 20. In particular, the event monitoring service 40 can **monitor the operational status** of nodes and packages within a cluster to report on the availability of the nodes, packages and of the cluster as a whole.

Detailed Description Text - DETX (26):
A cluster monitor 44a is a special type of monitor 44 designed to monitor cluster, node and package status on the cluster C. In the preferred embodiment, the cluster monitor 44a makes use of cluster MIBs (management information base) that are managed by daemons responsible for **tracking status** of nodes and packages. Generally, the node status is tracked with a heartbeat mechanism. Package status is tracked by the process or task ID assigned by the OS when the package is launched. Status changes to the cluster, node and package are reported in the form of an event to the EMS 40, which in turn provides the information onto the interested HA agent 20.

Detailed Description Text - DETX (59):
FIG. 7D illustrates a flowchart of a procedure performed by the HA agent 20 in response to a cluster event. In step 174, a cluster event is received from the cluster event monitor 44a. If the cluster event monitor 44a is not designed to send event messages when status changes, alternatively, the HA agent 20 could poll the cluster event **monitor 44a for status** changes. At step 176, the HA agent 20 prepares to package the cluster event in an availability or configuration event in accordance with the data types listed in Table II. In particular, a sequence number is generated for the event. At step 178, the status file 36 is updated with a current timestamp and the event is logged in the event log 32. At step 180, the event is transmitted to the HA server 22.

Detailed Description Text - DETX (66):
Next at step 208, the HA server 22 updates the configuration database 64, if necessary, according to the event received. The configuration database 64 **tracks the structure and status** of each of the monitored systems.

Additionally, the configuration database 64 is used by the HA server 22 to infer availability data for which no events were received (e.g., system downtime implies package downtime). At step 210, the procedure may generate a secondary event according to the following rules: system downtime implies node downtime; system downtime implies package downtime for all packages known to be running on the specified system; and system downtime implies cluster downtime if the given system was the sole remaining system in the cluster.

Claims Text - CLTX (1):

1. A fault tolerant method of monitoring one or more computers for availability, comprising: generating an event when a computer system **detects a change in its status** that affects availability; transmitting the event from the computer system to a central repository; and periodically re-transmitting the event if a receipt confirmation message is not received from the central repository.

Claims Text - CLTX (10):

10. A fault tolerant method of monitoring one or more computers for availability, comprising: generating an event containing a sequence number when a computer system **detects a change in its status** that effects availability; transmitting the event from the computer system to a central repository; comparing the sequence number of the event with a next expected sequence number computed from reading the central repository; and synchronizing events between the computer system and the central repository if the sequence number does not match the next expected sequence number.

US-PAT-NO: 6473857

DOCUMENT-IDENTIFIER: US 6473857 B1

TITLE: Centralized boot

---------- KWIC ---------

Abstract Text - ABTX (1):

The present invention provides a method for centralized and managed loading of **boot images** into one or more processors that are part of a file server for a mass storage system. In a computer system having at least one first controller, at least one input output processor (IOP), a first bus and a second bus, the present invention includes the steps of detecting readiness of the IOP to load a **boot image,** identifying across the first bus a location where the **boot image** will be loaded and loading the **boot image** across the second bus. The first controller may determine which of a plurality of **boot images** should be loaded. The first controller and the IOP may each have first and second processors, with communication between the first processors being across the first bus and **boot images** being accessed by the second processors across the second bus. On the IOP, the first processor may control **power** to the second processor and may **monitor the status** of the second processor, reporting across the first bus to the first controller's first processor regarding the status of the IOP's second processor. The **boot image** may be copied to memory local to the IOP's second processor or it may be made available across the second bus. By the method of this invention, the **boot image** supplied may be adapted to normal, diagnostic, crash dump or other purposes. The progress of IOP booting is tracked and monitored.

Brief Summary Text - BSTX (8):

The present invention provides a method for centralized and managed loading of **boot images** into one or more processors that are part of a file server for a mass storage system.

Brief Summary Text - BSTX (9):

In one embodiment, the present invention provides a method for booting processors in a computer system, the computer system including at least one first controller, at least one input output processor (IOP), a first bus and a second bus, including the steps of: detecting in the first controller readiness of the IOP to load a **boot image**; identifying across the first bus a location at which the **boot image** will be loaded; and loading the **boot image** across the second bus. It may additionally provide the steps of powering on the IOP, determining which of a plurality of **boot images** should be identified to the IOP, and booting the IOP from the **boot image**. One aspect of the invention may be that the detecting step includes signaling by the IOP across the first bus that it is ready to boot. Alternatively, the detecting step may include detecting that a flag has been set in the first controller's memory. The location at which the **boot image** will be loaded is memory on the IOP and either the IOP or the first controller may carry out the loading step by copying the **boot image** into the IOP's memory.

Brief Summary Text - BSTX (10):

In another embodiment, the present invention provides a method for booting processors in a computer system, the computer system including at least one first controller, at least one IOP, a first bus and a second bus, the first controller and the IOP having a first processor coupled to the first bus and a second processor coupled to both the first processor and the second bus, including the steps of: signaling from the IOP's first processor to the first controller's first processor that the IOP's second processor is ready to boot; identifying from the first controller's first processor to the IOP's first processor a location at which the IOP's second processor can access a **boot image**; and accessing from the IOP's second processor the **boot image**. This embodiment may additionally provide some or all of the steps of directing the IOP's first processor to power on the IOP's second processor, determining which

of a plurality of **boot images** should be identified by location to the IOP or its first processor, acknowledging by the receipt by the IOP's first and/or second processor of the location of the **boot image**; and directing the IOP's second processor, via the IOP's first processor, to access the **boot image**. As above, one aspect of the invention may be that the location at which the **boot image** can be accessed is memory on the IOP and either the IOP or the first controller may carry out the accessing step by copying the **boot image** across the second bus into the IOP's memory. In particular, the location may be memory of the IOP's second processor.

Brief Summary Text - BSTX (11):

An alternative embodiment of the present invention is a method for booting processors in a computer system, the computer system including at least one first controller, at least one input output processor (IOP), a first bus and a second bus, including the steps of: detecting in the first controller readiness of the IOP to load a **boot image**; identifying across the first bus a location at which the IOP can access the **boot image**; and accessing across the second bus the **boot image**. As above, this embodiment may additionally provide the steps of powering on the IOP, determining which of a plurality of **boot images** should be identified to the IOP, and booting the IOP from the **boot image**. Aspects of the detecting step, the **boot image** location and the loading step may be as summarized above.

Brief Summary Text - BSTX (12):

A further embodiment of the present invention is a method for booting processors in a computer system, the computer system including at least one first controller, at least one input output processor (IOP), a first bus and a second bus, the first controller and the IOP having a first processor coupled to the first bus and a second processor coupled to both the first processor and the second bus, including the steps of: signaling from the IOP's first processor to the first controller's first processor that the IOP's first and/or second processor is ready to boot; identifying from the first

US-PAT-NO: 6601165

DOCUMENT-IDENTIFIER: US 6601165 B2

TITLE: Apparatus and method for implementing **fault resilient booting** in a multi-processor system by using a flush command to control resetting of the processors and isolating failed processors

---------- KWIC ---------

TITLE - TI (1):
Apparatus and method for implementing **fault resilient booting** in a multi-processor system by using a flush command to control resetting of the processors and isolating failed processors

PAT-NO: JP02000215064A

DOCUMENT-IDENTIFIER: JP 2000215064 A

TITLE: **FAULT TOLERANT** METHOD OF HIGH RELIABILITY FILE SYSTEM
AND **BOOT** LOADING SYSTEM OF UNIX

PUBN-DATE: August 4, 2000

INVENTOR-INFORMATION:

NAME	COUNTRY
HANAYAMA, KOJI	N/A

ASSIGNEE-INFORMATION:

NAME	COUNTRY
NEC CORP	N/A

APPL-NO: JP11017613

APPL-DATE: January 26, 1999

INT-CL (IPC): G06F009/445

ABSTRACT:

PROBLEM TO BE SOLVED: To obtain a boot loading system which avoides a failure in loading a program without a trouble while eliminating the need for an input operation from an input device by providing a program loading means which loads a specified program from an external storage device to a storage device.

SOLUTION: On a data processor 2, a storage area managing means 21 takes program storage area information indicating an area where a program 13 is stored out of a program storage area 11 and stores it in an area storage memory part 31. A stored program managing means 22 takes program name information out of the external storage device 1 and stores it in the area storage memory part 31. The program loading means 23 loads programs 13 one by one from the external storage device 1 to a program storage part 32 in the storage device 2 according to the program storage area information and program name information

1 stored in the area storage memory part 31.

PUB-NO: GB002351820A

DOCUMENT-IDENTIFIER: GB 2351820 A

TITLE: **Fault resilient booting** in a multiprocessor environment

PUBN-DATE: January 10, 2001

INVENTOR-INFORMATION:

NAME	COUNTRY
MORRISON, JOHN A	US
ALLISON, MICHAEL S	US
EMBRY, LEO J	US
SILVA, STEPHEN J	US
FEEHRER, JOHN R	US

ASSIGNEE-INFORMATION:

NAME	COUNTRY
HEWLETT PACKARD CO	US

APPL-NO: GB00006154

APPL-DATE: March 14, 2000

PRIORITY-DATA: US27684699A (March 26, 1999)

INT-CL (IPC): G06F011/20

EUR-CL (EPC): G06F011/00

US-CL-CURRENT: 411/90, 453/54

DERWENT-ACC-NO: 1995-276329

DERWENT-WEEK: 199537

TITLE: **Fault tolerant** computer system with redundancy - has network of computers, one of which replaces working computer in case of failure by connecting to same data storage and re-**booting**

INVENTOR: VOGEL, P

PATENT-ASSIGNEE: MATTHIESEN SERVER SYSTEMS GMBH[MATTN]

PRIORITY-DATA: 1994DE-4443587 (December 8, 1994)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
DE 4443587 A1	August 10, 1995	N/A	006	G06F 011/20

APPLICATION-DATA:

PUB-NO	APPL-DESCRIPTOR	APPL-NO	APPL-DATE
DE 4443587A1	N/A	1994DE-4443587	December 8, 1994

INT-CL (IPC): G06F011/20, G06F015/163

ABSTRACTED-PUB-NO: DE 4443587A

BASIC-ABSTRACT:

The system has at least two computers. At any time, one computer is the working computer and the others are redundant. The interface of each computer (e.g. COM A, COM B COM C) is networked with all other computers, and is used to monitor the operational status of the computers. The HOST outputs of the computers controller cards (e.g. HOST A, HOST B, HOST C) are connected to a data storage (e.g. HD3), but only one computer is actively connected to the data storage at any time.

When a computer fails, this is detected by the network interface and the next redundant computer in hierarchical sequence takes its place. If the

working
computer fails, a redundant computer connects to the data store and re-boots as
the new working computer.

ADVANTAGE - Allows low cost construction of fault tolerant system, non-monolithic computer system located at different sites and using different
operating systems.

CHOSEN-DRAWING: Dwg.3/3

TITLE-TERMS: FAULT TOLERATE COMPUTER SYSTEM REDUNDANT NETWORK COMPUTER ONE
REPLACE WORK COMPUTER CASE FAIL CONNECT DATA STORAGE

DERWENT-CLASS: T01

EPI-CODES: T01-G03; T01-G05B; T01-M02A1;

SECONDARY-ACC-NO:
Non-CPI Secondary Accession Numbers: N1995-211316

ient processor **booting,** pre-failure ale

DERWENT-ACC-NO: 1995-276329

DERWENT-WEEK: 199537

TITLE: **Fault tolerant** computer system with redundancy - has network of computers, one of which replaces working computer in case of failure by connecting to same data storage and re-**booting**

INVENTOR: VOGEL, P

PATENT-ASSIGNEE: MATTHIESEN SERVER SYSTEMS GMBH[MATTN]

PRIORITY-DATA: 1994DE-4443587 (December 8, 1994)

PATENT-FAMILY:

PUB-NO	PUB-DATE	LANGUAGE	PAGES	MAIN-IPC
DE 4443587 A1	August 10, 1995	N/A	006	G06F 011/20

APPLICATION-DATA:

PUB-NO	APPL-DESCRIPTOR	APPL-NO	APPL-DATE
DE 4443587A1	N/A	1994DE-4443587	December 8, 1994

INT-CL (IPC): G06F011/20, G06F015/163

ABSTRACTED-PUB-NO: DE 4443587A

BASIC-ABSTRACT:

The system has at least two computers. At any time, one computer is the working computer and the others are redundant. The interface of each computer (e.g. COM A, COM B COM C) is networked with all other computers, and is used to **monitor the operational status** of the computers. The HOST outputs of the computers controller cards (e.g. HOST A, HOST B, HOST C) are connected to a data storage (e.g. HD3), but only one computer is actively connected to the data storage at any time.

When a computer fails, this is detected by the network interface and the next redundant computer in hierarchical sequence takes its place. If the

working
computer fails, a redundant computer connects to the data store and re-boots as
the new working computer.

ADVANTAGE - Allows low cost construction of fault tolerant system, non-monolithic computer system located at different sites and using different
operating systems.

CHOSEN-DRAWING: Dwg.3/3

TITLE-TERMS: FAULT TOLERATE COMPUTER SYSTEM REDUNDANT NETWORK COMPUTER ONE
REPLACE WORK COMPUTER CASE FAIL CONNECT DATA STORAGE

DERWENT-CLASS: T01

EPI-CODES: T01-G03; T01-G05B; T01-M02A1;

SECONDARY-ACC-NO:
Non-CPI Secondary Accession Numbers: N1995-211316

US-PAT-NO: 6421777

DOCUMENT-IDENTIFIER: US 6421777 B1
See image for Certificate of Correction

TITLE: Method and apparatus for managing boot images in a distributed data processing system

---------- KWIC ---------

Detailed Description Text - DETX (28):
Although the depicted examples illustrate the use of the processes of the present invention to install a new or upgraded operating system, the processes of the present invention also may be applied to the sending of any combination of images, including more than just the boot image, down to a target client. For example, an initial BIOS Flash Utility may be downloaded as the boot image, followed by reboot. During the reboot, a DOS image with a hardware scan program may be downloaded followed by reboot. When this second reboot occurs, an OS/2 based installer image may be then downloaded followed by a reboot in which a **local boot image** is then downloaded. As can be seen, the present invention allows a wide ranging set of maintenance tasks on a target client computer. An installer image contains a full DOS environment with which the install program is run. The **local boot image** is a set of instructions to load the boot sector of the hard disk.

Detailed Description Text - DETX (43):
A server state daemon service (STATEDMN.EXE) is employed to monitor the state of the client's boot phases and to switch from remote boot to local boot. Once the operating system is installed on the client's hard drive, this daemon checks the state file and updates the remote boot file client's entry to LOCAL image from REMOTE image. The **LOCAL boot image** reboots the clients from the local hard drive.

Claims Text - CLTX (2):

2. The method of claim 1, wherein the set of boot images includes a **local boot image,** wherein sending of the **local boot image** to the client data processing system causes the client data processing system to boot from a local image located within the client data processing system.

Claims Text - CLTX (3):
3. The method of claim 1, wherein the **local boot image** is sent to the client data processing system when the state of the client data processing system is local.

Claims Text - CLTX (10):
10. The method of claim 8, wherein the set of boot images includes a **local boot image,** wherein sending the **local boot image** to the client data processing system causes the client data processing system to boot from a local operating system located within the client data processing system.

Claims Text - CLTX (17):
17. The server data processing system of claim 16, wherein the set of boot images includes a **local boot image,** wherein sending of the **local boot image** to the client data processing system causes the client data processing system to boot from a local image located within the client data processing system.

Claims Text - CLTX (18):
18. The server data processing system of claim 16, wherein the **local boot image** is sent to the client data processing system when the state of the client data processing system is local.

Claims Text - CLTX (25):
25. The data processing system of claim 23, wherein the set of boot images includes a **local boot image,** wherein sending the **local boot image** to the client data processing system causes the client to boot from a local operating system located within the client data processing system.

Current US Original Classification - CCOR (1):
713/2

US-PAT-NO: 6308265

DOCUMENT-IDENTIFIER: US 6308265 B1

TITLE: Protection of boot block code while allowing write accesses to the boot block

---------- KWIC ---------

Detailed Description Text - DETX (11):
If the validation has been confirmed, then, in a third step 320, a flag is set to provide an indication that the boot block (first region) is being updated. Such a flag is called a boot-block-in-progress flag in the present invention, and may be a hardware sticky bit or the like. The boot-block-in-progress flag provides an indication that an updating of the BIOS is being performed. Such a flag may be set by way of a bit being set in a latch or flip-flop, for example, whereby if a power failure occurs, that bit would still be in the set state upon power up or reset of the PC. If a reset or power up occurs with the flag being in the set state, the CPU will boot the PC from the backup boot block image in the **second region, and not from the boot block image** in the first region.

Current US Original Classification - CCOR (1):
713/2

Current US Cross Reference Classification - CCXR (3):
714/6

US-PAT-NO: 6101601

DOCUMENT-IDENTIFIER: US 6101601 A

TITLE: Method and apparatus for hibernation within a distributed data processing system

---------- KWIC ---------

Detailed Description Text - DETX (8):
In the depicted example, if this information is missing from server 300, NC 310 will then perform a normal remote boot, downloading the boot image and other information from server 300. Alternatively, information may be present that affirmatively instructs NC 310 to perform a normal remote boot. A normal boot is typically performed because the **local boot image** is outdated or the server wants to rehibernate. In addition, either or both boot images 302-308 and NC file 316 may be stored elsewhere in the distributed data processing system, such as a storage unit (i.e. storage unit 106 in FIG. 1). This process allows the server to maintain client management of client NCs. In addition, the load of the image is fast because all of the images are local and the machine state is predetermined in the hibernation image (i.e. the applications and operating system that are in the image have already been initialized). Depending upon the implementation, incremental hibernation may occur. An instance may occur in which part of the image on the local NC is outdated, but most of the information is still valid. In this case, a server may send a message indicating that portions of the **local boot image** may be used except for a new portion, which is sent with the message or referenced by the message.

Current US Original Classification - CCOR (1):
713/2

US-PAT-NO: 5948101

DOCUMENT-IDENTIFIER: US 5948101 A

TITLE: Methods and systems for booting a computer in a distributed computing system

---------- KWIC ---------

Claims Text - CLTX (28):
(c) transmitting a boot image request over said network to facilitate the
receipt of a boot image for said computer, said boot image request including
said **second identifier, said boot image** corresponding to said second identifier.

Current US Original Classification - CCOR (1):
713/2

US-PAT-NO: 5694583

DOCUMENT-IDENTIFIER: US 5694583 A

TITLE: BIOS emulation parameter preservation across computer bootstrapping

---------- KWIC ---------

Claims Text - CLTX (24):

7. The method of claim 6, wherein the CD-ROM configuration includes a drive emulation having a booting catalog pointing to at least one default bootable disk image and another entry, wherein the booting catalog also points to a **second bootable disk image,** and wherein, upon the command for warm boot, an initial boot sequence is initiated from the booting catalog, and thereafter, selectively, from the another entry in the booting catalog.

Current US Cross Reference Classification - CCXR (1):

713/2

US-PAT-NO: 5452454

DOCUMENT-IDENTIFIER: US 5452454 A

TITLE: Generic remote boot for networked workstations by creating **local bootable code image**

---------- KWIC ---------

TITLE - TI (1):
Generic remote boot for networked workstations by creating **local bootable code image**

Drawing Description Text - DRTX (12):
FIG. 14 is a memory map diagram showing the state of the PC client memory just **prior to booting from the LAD disk image** during the generic remote boot procedure of this invention.

Claims Text - CLTX (6):
copying an image of the bootable code over the network to the local data processing system to create a **local bootable code image** stored in the local data processing system, the bootable code including an operating system; and

Claims Text - CLTX (8):
booting the local data processing system from the **local bootable code image** stored in the local data processing system and loading the operating system while not protecting a memory space containing the task image code module to thereby leave the memory space available to the operating system.

Claims Text - CLTX (46):
the step of booting the data processing system from the **local bootable code image** further comprises

Claims Text - CLTX (150):
copying means for copying an image of the bootable code over the network to the local data processing system to create a **local bootable code image** stored

in the local data processing system, the bootable code including an operating
system; and

Claims Text - CLTX (151):
booting means for booting the local data processing system from the **local bootable code image** after it has been completely stored in the local data
processing system while the network communications link is inactivated, the
booting means loading the operating system while not protecting a memory space
containing the task image code module to thereby leave the memory space available to the operating system.

Claims Text - CLTX (200):
copying means for copying an image of the bootable code over the network to
the workstation to create a **local bootable code image** stored in the workstation, the bootable code including an operating system; and

Claims Text - CLTX (201):
booting means for booting the workstation from the **local bootable code image**
only after the bootable code includinq the operating system has been stored in
the workstation and while the network link is inactivated, the booting means
loading the operating system while not protecting a memory space containing the
task image code module to thereby leave the memory space available to the
operating system.

Current US Original Classification - CCOR (1):
713/2

US-PAT-NO: 5379431

DOCUMENT-IDENTIFIER: US 5379431 A

TITLE: Boot framework architecture for dynamic staged initial program load

---------- KWIC ---------

Detailed Description Text - DETX (266):

Primary objects are those boot image based objects which the BootConductor instantiates. These objects are fundamental to the BootConductor's focal job of getting the system up to mount the OPD. An example of a primary object is TMachine. Secondary objects are those boot image based objects which are not primary objects, but do want to run at boot time. These secondary objects can be mandatory or optional. An example of an optional secondary object would be some background printer status server. The boot conductor gives secondary objects several opportunities to execute.

Current US Cross Reference Classification - CCXR (1):

713/2

US-PAT-NO: 6374366

DOCUMENT-IDENTIFIER: US 6374366 B1

TITLE: Automated drive repair systems and methods

---------- KWIC ---------

Detailed Description Text - DETX (63):
The interface of the drive repair suite 324 is configured to export one method. This method is a SystemDiagnose method that is configured to analyze the system (i.e., the primary drive system) status, and then suggest an **initial** action to be taken to repair the **detected problematic status** of the system. The recommended action to be taken is in the form of a method which is part of a state machine that has a plurality of methods (e.g., Action 1-Action 10). Each method, therefore, represents an action that may be taken to fix the detected problem. As used herein, an Action is equivalent to a method. Just as the SystemDiagnose method returns a suggested next action, each individual action will return a next suggested action to be taken to repair the detected problem.

Detailed Description Text - DETX (149):
Although the foregoing invention has been described in some detail for purposes of clarity of understanding, it will be apparent that certain changes and modifications may be practiced within the scope of the appended claims. For example, although a preferred type of peripheral storage device is a Jaz.TM. drive, any type of peripheral storage devices or built-in computer storage devices can be used. In addition, the storage devices can be either physically located next to the computer system itself or **remotely** networked over, e.g., a local area network (LAN) or the Internet.

Detailed Description Paragraph Table - DETL (1):
TABLE 1 ABS MBR Layout 0 1 2 3 4 5 6 7+ [MBR] Genesis ABS 1 Signature ABS 2 Image or **Boot Image** [Boot Signature Management Code part 1]

[Boot
Management Code part 2--overflow from part 1 if needed]

US-PAT-NO: 5832222

DOCUMENT-IDENTIFIER: US 5832222 A

TITLE: Apparatus for providing a single image of an I/O subsystem in a geographically dispersed computer system

---------- KWIC ---------

Brief Summary Text - BSTX (4):
The prior art, in an attempt to transcend the physical limitations of accessing common data from clustered computer systems, has attempted to expand disk capacity outside of a single box (e.g. disk drive unit). For instance, it is known to **remotely** access a peripheral such as a disk associated with a **remote** computer from a local computer. In particular, a known software product such as **Remote** File Sharing software (RFS) for UNIX permits a local device or node to gain access to a peripheral on another computer system. However, RFS is specialized software that tends to be unreliable and slow, and more importantly is not transparent to the local or to the **remote** computer system.

Brief Summary Text - BSTX (12):
Pursuant to yet another embodiment of the present invention, there is provided a computer system including a local processing unit having a first operating system executing thereon, a storage device for storing data, a physical device driver for facilitating the transfer of the data between the local processing unit and the storage device, and a distributed data module transparently interfaced to the operating system and the physical device driver for replicating the data across a network to a first **remote** processing unit.

Drawing Description Text - DRTX (4):
FIG. 3 is a block diagram of a communication architecture for permitting transparent data replication between a local processing unit and one or more geographically **remote** processing units of the computer system shown in FIG. 2;

Detailed Description Text - DETX (6):
In addition, the middleware 8 acts as a switch for data that is to be written to a local peripheral via the physical device driver 5. In particular, the middleware 8 can replicate data to, or read data from a **remote** peripheral on path 9 without affecting the performance of the I/O subsystem interface 4 or the physical device driver 5. In the embodiment being described, the middleware 8 resides on a host system as described further below. However, it should be appreciated that the middleware 8 may reside on a controller board such as a host adapter, or reside at a RAID subsystem level in order to replicate data in the same manner as set forth below with respect to a host system.

Detailed Description Text - DETX (7):
Referring now to FIG. 2, there is shown a block diagram of multi-node computer system 10 which incorporates the features of the present invention therein. The multi-node computer system 10 includes a number of geographically dispersed processing units or nodes 12 such as host computers. Two processing units, a local processing unit 12a and a geographically **remote** processing unit 12b are shown in FIG. 2 to facilitate the explanation of the computer system 10. However, it should be appreciated that any number of additional local and/or **remote** processing units may be added to the computer system 10 without departing from the scope of the invention described hereafter.

Detailed Description Text - DETX (13):
In the embodiment being described, the switch-over software 24 executing on local processing unit 12a may **monitor the operational status of remote** processing unit 12b across the network 14 to insure that processing unit 12b remains **powered** up. Likewise, the switch-over software 24 executing on **remote** processing unit 12b may **monitor the operational status** of local processing unit 12a across the network 14 to insure that processing unit 12a remains **powered** up. The switch-over software 24 monitors the respective processing units 12 by a conventional heartbeat mechanism that is transmitted and received across the

network 14. A suitable switch-over software package is the LifeKeeper software
package available from AT&T.

Detailed Description Text - DETX (15):
By way of example, if the switch-over software 24 resident on the **remote**
processing unit 12b detects that the local processing unit 12a has failed in
some respect, such as the processing unit 12a fails to respond to queries
across the network 14, then the switch-over software 24 resident on processing
unit 12b will mount the software application that was previously executing on
processing unit 12a, onto processing unit 12b. Once the software application
is mounted on processing unit 12b, processing of the same data can begin again
because the data was previously automatically replicated across the network 14
from the disk 32a of processing unit 12a to the disk 32b of processing unit 12b
by the distributed data model software 26 as described further below. The
distributed data model software 26 replicates the data from the disk 32a to
disk 32b in the same order as the data is written to disk 32a.

Detailed Description Text - DETX (16):
Thus, in a switch-over application, one processing unit 12 runs an application while the other processing unit 12 is updated or otherwise has data
replicated thereto. In particular, a distributed data configuration consists
of two pieces or disks, the local piece being the disk 32 of the processing
unit 12a and the **remote** piece being the disk 32 of the processing unit 12b.

Detailed Description Text - DETX (17):
When an application runs on processing unit 12a, the data is duplicated to
the disk 32b of the **remote** processing unit 12b. If a situation occurs where
the application running on processing unit 12a fails or is otherwise interrupted, then the processing unit 12b boots up a copy of the application on
disk 32b, and the disks 32 comprising the distributed mirror reverse roles.
That is, the local piece of the distributed data configuration becomes disk 32b
and the **remote** piece of the distributed data configuration becomes 32a.

Detailed Description Text - DETX (19):
The parallel software application 22, when used in conjunction with the distributed data model software 26, provides a single system data image to the geographically dispersed processing units 12 of the computer system 10. That is, the parallel software applications execute simultaneously on processing units 12. The distributed data model software 26 permits data that is written to the disk 32 of the local processing unit 12a to be automatically replicated the disk 32 of one or more **remote** processing units 12.

Detailed Description Text - DETX (20):
Likewise, the distributed data model software 26 permits data that is written to the disk 32 of the **remote** processing unit 12b to be automatically replicated to the disk 32 of the local processing unit 12a. Thus, the distributed data model software 26 permits the processing units 12 to have a single system image of I/O subsystem (e.g. a signal system image of the data on the disks 32) even though the processing units 12 are geographically separated.

Detailed Description Text - DETX (21):
The distributed data model software 26 utilizes a conventional I/O path 36 (FIG. 2) for facilitating I/O transfers (e.g. I/O reads and writes) to the local disk 32. In addition, the distributed data model software 26 includes a second I/O path 38 for facilitating the transparent (e.g. without the knowledge of the local processing unit 12) replication of data bound for a local disk 32, across the network 14 to a **remote** disk associated with a **remote** processing unit as described further below.

Detailed Description Text - DETX (22):
Thus, in a parallel application, both processing units 12 run an application at the same time on a shared distributed data configuration. In the shared distributed data configuration, the processing unit 12a has the disk 32a configured as the local piece and the disk 32b configured as the **remote** piece of the distributed shared data configuration. Simultaneously, the processing unit 12b has disk 32a configured as the **remote** piece and the disk 32b

configured as the local piece of the distributed shared data configuration. In parallel applications, the processing units 12a, 12b utilize synchronous data replication when writing data to the disks 32a, 32b.

Detailed Description Text - DETX (23):

Referring now to FIG. 3, there is shown an architecture 39 of the distributed data model software 26 that is resident on processing unit 12a, and an architecture 86 of the distributed data model software 26 that is resident on the processing unit 12b. In particular, the architectures 39, 86 shown in FIG. 3 and described in detail hereafter, facilitate the transparent replication of data along the second I/O path 38 (FIG. 2) from a local processing unit 12a to a sequential access device (e.g. tape drive) or a random access device (e.g. hard disk, optical disk, hardware RAID subsystem, software RAID) associated with one or more **remote** processing units 12b.

Detailed Description Text - DETX (24):

In addition, the architectures 39, 86 shown in FIG. 3 and described in detail hereafter, facilitate sending state information along the second I/O path 38 from the local processing unit 12a to one or more **remote** processing units 12b. Note that the architecture 39 also facilitates transferring I/O requests (e.g., I/O reads and writes) between the local processing unit 12a and the local disk 32 along the conventional I/O path 36 in a manner well known to one of ordinary skill in the art. Thus, further discussion thereof is not warranted.

Detailed Description Text - DETX (31):

The distributed storage module 46 and the communication infrastructure 49 cooperate to form the distributed data model software 26 (FIG. 2). As shown in FIG. 4 and described in detail below, the distributed data model software 26 includes a number of subroutines, namely an entry point routine 26a, an acknowledge routine 26b, an done routine 26c, a send routine 26d, and a time out routine 26e. It should be appreciated that the distributed storage module 46 and communication infrastructure 49 are also resident on the **remote** processing unit 12b. In addition, it should be appreciated that some of the subroutines mentioned above are executed on a local processing unit and

other
subroutines are executed on a **remote** processing unit as described further
below.

Detailed Description Text - DETX (34):
Referring now to FIG. 6, a DPDU data structure 62 includes a number of
fields such as a request type field 64. The request type field 64 identifies
whether state information or I/O data is to be transferred to the **remote**
processing unit 12. If state information is to be broadcast to the processing
unit 12b, the field 64 may identify a set or a clear state information
operation. If I/O data is to be transferred to the **remote** processing unit 12b,
the field 64 identifies either a read I/O or write I/O request.

Detailed Description Text - DETX (35):
The DPDU data structure 62 may also include (1) a data address field 66
which links the completion of the request on the **remote** processing unit 12b to
the original request from the kernel, e.g. I/O request, (2) a memory release
field 67 which, when set, instructs the transport module 50 to release the
memory reserved for the DPDU 62 once the packet is successfully sent to the
target processing unit 12b, (3) a target processing unit identification field
68 which identifies which processing unit that the request should be directed
to, (4) a source processing unit identification field 69 which identifies which
processing unit that the request was sent from, and (5) a data field 70 which
contains the data and/or state information to be sent or broadcast to the
target processing unit(s) 12 in the computer system 10. The DPDU data
structure 62 may also include a control mode field 71 which is active only when
the TPDU data field 70 contains state information as indicated by the request
type field 64.

Detailed Description Text - DETX (37):
A TPDU data structure 72 includes a path number field 74 which identifies
the preferred network path for communicating the packet to the **remote**
processing unit(s) 12b, a sequence number field 76 which guarantees the
processing order of the requests by the **remote** processing unit 12b, and a DPDU
field 78 which encapsulates the DPDU data structure 62.

Detailed Description Text - DETX (39):

Referring now back to FIG. 3, the known kernel I/O driver(s) 44 pass conventional data blocks to the distributed storage module 46. By conventional, it is meant that the kernel I/O driver(s) 44 pass to the distributed storage module 46, buffer headers that would normally be passed from the operating system to the physical device driver 28 were it not for the presence of the distributed data model software 46. The buffer header typically contains information such as the block number on the **remote** disk 32 needed for an I/O request. The kernel I/O driver(s) 44 also write to the local disk 32 along the conventional path 36 (FIG. 1) as previously described. The kernel I/O driver(s) 44 may be provided as part of the operating system of the computer system 10, or may provided as part of an add-on volume management software package such as Disk Array PLUS or Tape Array PLUS volume management software available from AT&T.

Detailed Description Text - DETX (40):

After the kernel I/O driver(s) 44 pass the buffer header to the distributed storage module 46, the entry point routine 26a (FIG. 4) of the distributed data model software 26 allocates and initializes a buffer (not shown) from a pool of available memory (not shown). The buffer is used for building the DPDU 62. Preferably, the volume management software may allocate the buffer for the DPDU 62, which buffer should be large enough to accommodate the encapsulation requirements of the TPDU 72 and the NPDU 80 (this is the case for sending state change information to the **remote** processing unit). The initialization of the DPDU 62 by the entry point routine 26a includes the tasks of: (1) indicating if the request is an I/O read or write request, or a set or clear state information request; (2) placing the address of the buffer header that is passed from the kernel I/O driver(s) 44 onto the outstanding request queue 48 for use in completing the particular request as described further below; and (3) identifying the **remote** processing unit 12b to receive the request.

Detailed Description Text - DETX (41):

With regard to the third initialization step above, the distributed storage

module 46 may replicate or broadcast to one or more **remote** geographically dispersed target processing units 12b. The transport module 50 encapsulates the DPDU 62 into a TPDU 78 and facilitates sending the TPDU 78 across the network 14 to the target **remote** processing unit 12b.

Detailed Description Text - DETX (42):
In addition to initializing the DPDU 62, the entry point routine 26a provides system I/O flow-control by monitoring the outstanding requests on the outstanding request queue 48 to see if a predetermined system threshold has been met. More specifically, there is a threshold on the amount of memory that can be allocated for DPDUs 62. If too many DPDUs 62 are outstanding, then it is possible that threshold may be reached. In order to avoid reaching the threshold, the entry point routine 26a governs the rate of allocation of DPDUs 62 by maintaining an ordered work list, keeping track of the number of outstanding requests, and deciding if the requests can be sent to the **remote** site.

Detailed Description Text - DETX (43):
The distributed storage module 46 passes the DPDU 62 to the send routine 26d for encapsulation into the DPDU field 78 of TPDU 72. The send routine 26d also encodes the second tier encapsulation information into the header of the TPDU 72. This includes encoding a sequence number into the sequence number field 76 of the TPDU 72 so that the packet (e.g. TPDU) is delivered to the **remote** site, in order, on the path specified in the path number field 74 of the TPDU 72. The TPDU 72 is then placed on the transmit queue 52 for use in subsequent retransmissions of the TPDU 72, if necessary. That is, the transmit queue 52 is used by the transport module 50 to guarantee the sequence that packets are send across the network 14.

Detailed Description Text - DETX (44):
The transport module 50 also passes the TPDU 72 to the kernel network device driver 54 via system interfaces to transmit the TPDU 72 to the **remote** processing unit 12b. The kernel network device driver 54 encapsulates the TPDU 72 into the TPDU field 84 of the NPDU 80 which is then conveyed to the

remote
processing unit 12b using the protocol and methods defined by the known low-level communication driver (e.g. UDP). The low-level communication driver
sends the NPDU packet 80 over the network 14 independent of the distributed
data model software 26.

Detailed Description Text - DETX (45):
The transport module 50 receives TPDU packets 72 from the **remote** processing
unit 12b. The incoming TPDU packets 72 may contain acknowledgment information
regarding a previous TPDU packet 72 sent to the **remote** processing unit 12b, or
may contain data to be replicated onto the local disk 32 from the **remote**
processing unit 12b during an active-active mode of operation. The transport
interrupt routine 56 is called by a network device interrupt routine or low-level communication driver interrupt routine (not shown) when an incoming
packet is received. The transport interrupt routine 56 places the incoming
TPDU 72 on the receive queue 58 ordered according to the assigned sequence
number stored in the sequence number field 76 of each TPDU 72. The lowest
sequence numbered TPDU packet 72 is placed at the front of the receive queue
58, and the highest numbered TPDU packet 72 is placed at the rear of the
receive queue 58.

Detailed Description Text - DETX (49):
The distributed storage module 46 handles acknowledgments. The acknowledgment routine 26b (FIG. 5) is called by the receive daemon 60 when
requests return from the **remote** processing unit 12b. The acknowledgment
routine 26b removes the relevant buffer header from the outstanding request
queue 48. The distributed storage module 46 obtains status information from
the TPDU 72 regarding the pending request and sets any error fields, if necessary, in the buffer. The distributed storage module 46 obtains the
address of the buffer from the received TPDU 72 and passes the address to the
done routine 26c.

Detailed Description Text - DETX (51):
The time-out routine 26e (FIG. 5) is initialized when the first **remote** I/O
operation is performed on a configured **remote** device (i.e. replicating

data to
the disk of the **remote** processing unit 12b). At pre-determined specified
intervals, the time-out routine 26e checks the DPDUs 62 on the outstanding
request queue 48. DPDU packets 62 found that exceed the allotted time to
complete will have their error fields (not shown) set in the buffer and be
removed from the outstanding request queue 48. The failed buffer will then be
passed to the done routine 26c. In addition, an appropriate error may be
displayed on a console.

Detailed Description Text - DETX (53):
Returning again to FIG. 3, the architecture 86 for the **remote** processing
unit 12b includes a number of routines that reside in a kernel layer 88 and in
an interrupt layer 90 of the operating system. In particular, the architecture
86 for the **remote** processing unit 12b includes a communication infrastructure
91 comprising a transport interrupt routine 92 that resides in the interrupt
layer 90, and a receive queue 94, a receive daemon 96, a transport module 110
and a kernel network device driver 112 that reside in the kernel layer 88. The
architecture 86 also includes a distributed storage module 98, one or more I/O
queues 100, one or more I/O daemons 102, and one or more kernel I/O drivers 104
that reside in the kernel layer 90, and a kernel done routine 106 and a DSM
callback routine 108 that both reside in the interrupt layer 88. As with the
communication infrastructure 49, the communication infrastructure 91 may be any
known communication infrastructure which complies with the known ISO/OSI
(International Organization for Standardization Open Systems Interconnection)
model.

Detailed Description Text - DETX (58):
Once preliminary validation of the received DPDU 62 has been completed, the
entry point routine 26a allocates a system buffer or control structure and
initializes the buffer to include the DSM callback routine 108, and include all
other information from the DPDU 62 to complete the **remote** request. The DSM
callback routine 108 permits the distributed storage module 98 to be

notified
by a virtual software management layer (when performing I/O to a virtual disk)
or a physical device driver (when performing disk I/O) when the **remote** request
is completed.

Detailed Description Text - DETX (60):
As previously mentioned, the kernel I/O drivers 104 may be add-on volume
management software such as Disk Array PLUS or Tape Array PLUS volume
management software provided by AT&T. For I/O requests, the kernel I/O drivers
104 translate the I/O request into a conventional physical I/O request for use
in reading or writing to the peripheral associated with the **remote** processing
unit 12b. That is, the physical I/O request is then serviced by the physical
device driver 28 (FIG. 1) which reads/writes from a particular block on the
disk 32 in a conventional manner.

Detailed Description Text - DETX (62):
At the interrupt level 90, when a physical I/O request completes, the kernel
done routine 106, calls the DSM callback routine 108. The DSM callback routine
108 may also be called by a volume management software routine (not shown)
using standard kernel callback conventions if the volume management software is
resident on the computer system 10. In the later case, the volume management
software routine will be called by the kernel done routine 106 when the
physical I/O request completes. Thus, the DSM callback routine 108 is invoked
when all requests have been completed on the **remote** processing unit 12b for the
particular **remote** request. The DSM callback routine 108 marks the completed
remote request in the corresponding buffer that is on one of the I/O queue(s)
100 and notifies the I/O daemon(s) 102.

Detailed Description Text - DETX (68):
Thus, the application shown in FIG. 7 provides for a copy of the data to be
available at a centralized server 200 with the data being as accurate as of the
last restore operation. If the data is to be backed up to tape as well, then
backup operations to a tape can be done on the backup server 200 with no impact
to the users or applications running on the processing units 202-206,

even if
the tape backup is performed during peak hours of the day.
Alternatively, the
server 200 may run an application locally, and may need to update
identical
databases resident on each of the **remote** processing units 202-206. As
such,
the server 200 could operate in a branch office and replicate
information to
the processing units 202-206 in **remote** branch offices in the same
manner as
discussed above.

Detailed Description Text - DETX (69):
FIG. 8 is a block diagram showing the distributed data model
software of the
present invention being used for replicating a boot device across a
network.
In particular, the communication architecture provides the capability
to
replicate the /root, /usr, /var, and /stand files of a local system 208
to a
remote system 210 (along with data files on other disks of the local
system
208) when the distributed data model software 26 of the present
invention is
present on both systems. In particular, the local system 208 includes
a known
volume management software layer or product 212, and two disks, namely,
a boot
disk 214 and a secondary boot disk 216. The secondary boot disk 216
stores a
copy of the boot disk 214 locally. It is known in the art to use the
volume
management product 212 to mirror a copy of the boot disk 214 to the
secondary
boot disk 216.

Detailed Description Text - DETX (70):
The local system 208 also includes a distributed data configuration
software
layer or product 218 inserted below the local volume management layer
212. It
should be appreciated that the distributed data configuration product
218 may
represent the distributed data model software 26 of the present
invention. In
this configuration, the distributed data configuration product 218
transparently (without impacting or intruding on operating system
software or
the volume management software) mirrors the /root, /usr, /var, and
/stand files
not only to the secondary boot disk 216 but also to a **remote** disk 220
associated with the **remote** system 210. Thus, if the local system 208
fails,
such as due to a catastrophe, the **remote** system 210 can be booted-up

from the
remote disk 220 which is a copy of the boot disk 214 of the local system 208.

Detailed Description Text - DETX (71):
When the **remote** system 210 comes up, it is using the local system's configuration and data, and the **remote** system 210 **boots up the image** that was being executed on the local system 208. That is, the **remote** system 210 boots up as if it were the local system 208. Since the **remote** system 210 has all of the local system's data disks and program disks, the applications which were running on the local system 208 can be restarted on the **remote** system 210 without having any knowledge that they are now on a different system. It should be appreciated that the boot disk 214, secondary boot disk 216 and **remote** disk 210 may be a hard disks, optical disks, hardware RAID sub-systems, or software RAID sub-systems.

Detailed Description Text - DETX (78):
For example, the described invention provides a distributed data model (data over a network) which permits a cluster of nodes or processing units to access common data over a network. The described invention permits clustering of processing units across different hardware platforms and I/O subsystems. The described invention manages multiple I/O daemons on a **remote** processing unit to maximize I/O throughput performance at a **remote** site.

Detailed Description Text - DETX (79):
The described invention permits **remote** I/O completion acknowledgments to be piggy-backed to minimize interconnect and system bus bandwidth utilization. The described invention synchronizes **remote** requests on the **remote** processing unit in the same order presented to the I/O subsystem on the local processing unit independent of the number of paths interconnecting the nodes or processing units.

Detailed Description Text - DETX (80):
The described invention permits continuous data service in the event of a node or processing unit failure by permitting the application to be

switched
over to a **remote** processing unit. The described invention permits two or more
geographically dispersed processing units access to common or replicated data.
The described invention is functionally transparent to the system and to the
applications running on the system. The described invention provides synchronous and asynchronous data replication services that can be used simultaneously.

Claims Text - CLTX (60):
a distributed data module transparently interfaced to said operating system
and said physical device driver for replicating said data across a network to a
remote processing unit, wherein said distributed data module includes:

Claims Text - CLTX (63):
a network device driver for transferring said second protocol unit across
said network to said **remote** processing unit; and

Claims Text - CLTX (67):
14. The computer system of claim 13, wherein said distributed data module
replicates said data across said network to a plurality of **remote** processing
units.